What are base modules?

The base modules offer directives that allow you to define the parameters of the basic functionality of Nginx. They cannot be disabled at compile time, and as a result, the directives and blocks that they offer are always available. Three base modules have been distinguished:

- **Core module**: Consists of essential features and directives such as process management and security
- **Events module**: Lets you configure the inner mechanisms of the networking capabilities
- **Configuration module**: Enables the inclusion mechanism

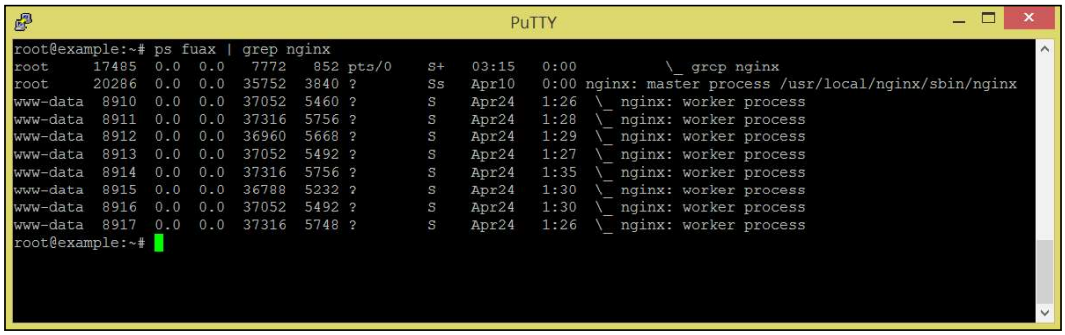
These modules offer a large range of directives; we will be detailing them individually with their syntaxes and default values.

The Nginx process architecture

Before we start detailing the basic configuration directives, it is necessary to understand the overall process architecture, that is, the way that the Nginx daemon works behind the scenes. Although the application comes as a simple binary file (and a somewhat lightweight background process), the way it functions at runtime can be relatively complex.

A unique process—the *Master Process*—exists in memory from the very moment that Nginx starts. It is launched with the current user and group permissions—usually root/root if the service is launched at boot time by an `init` script. The master process itself does not process any client request; instead, it spawns the processes that do, that is, the *Worker Processes*, which are affected to a customizable user and group.

From the configuration file, you can define the number of worker processes, the maximum connections per worker process, the user and group that the worker processes are running under, and more. The following screenshot shows an example of a running instance of Nginx with eight worker processes running under the `www-data` user account.



Core module directives

The following is the list of directives made available by the core module. Most of these directives must be placed at the root of the configuration file, and can only be used once. However, some of them are valid in multiple contexts. If that is the case, the following is the list of valid contexts under the directive name:

Name and context	Syntax and description
`daemon`	Accepted values: `on` or `off` Syntax: `daemon on;` Default value: `on` Enables or disables daemon mode. If you disable it, the program will not be started in the background; it will stay in the foreground when launched from the shell. This may come in handy for debugging, in situations where you need to know what causes Nginx to crash and when.
`debug_points`	Accepted values: `stop` or `abort` Syntax: `debug_points stop;` Default value: `None` Activates debug points in Nginx. Use `stop` to interrupt the application when a debug point comes about in order to attach a debugger. Use `abort` to abort the debug point and create a core dump file. To disable this feature, simply do not use the directive.

Name and context	Syntax and description

`env`	Syntax:
	`env MY_VARIABLE;`
	`env MY_VARIABLE=my_value;`
	Allows you to define or redefine environment variables.

`error_log`
Context: `main`, `http`, `server`, and `location`
	Syntax:
	`error_log /file/path level;`
	Default value: `logs/error.log error`.
	Where `level` is one of the following values: `debug`, `info`, `notice`, `warn`, `error`, `crit, alert, emerg` (from the most to least detailed: `debug` provides frequent log entries, `emerg` reports only the most critical errors).
	Enables error logging at different levels: Application, HTTP server, virtual host, and virtual host directory.
	By redirecting the log output to `/dev/null`, you can disable error logging. Use the following directive at the root of the configuration file:
	`error_log /dev/null crit;`
	Instead of specifying a file path, you might also select one of the following alternatives: `stderr` will send log entries to the standard error file, `syslog` to the system log, and `memory` will store the log entries in the memory.

`lock_file`	Syntax: File path `lock_file logs/nginx.lock;` Default value: Defined at compile time Use a lock file for mutual exclusion. This is disabled by default, unless you enabled it at compile time. On most operating systems, locks are implemented using atomic operations, so this directive is ignored anyway.
`log_not_found` Context: `main`, `http`, `server`, and `location`	Accepted values: `on` or `off` `log_not_found on;` Default value: `on` Enables or disables the logging of **404 not found** HTTP errors. If your logs get filled with 404 errors due to missing `favicon.ico` or `robots.txt` files, you might want to turn this off.

Name and context	Syntax and description
`master_process`	Accepted values: `on` or `off` `master_process on;` Default value: `on` If enabled, Nginx will start multiple processes: a main process (the master process) and worker processes. If disabled, Nginx works with a unique process. This directive should be used for testing purposes only, as it disables the master process—thus, clients will not be able to connect to your server.
`pcre_jit`	Accepted values: `on` or `off` `pcre_jit on;` Enables or disables the Just-In-Time compilation for regular expressions (PCRE from version 8.20 and above), which may speed up their processing significantly. For this to work, the PCRE libraries on your system must be specifically built with the `--enable-jit` configuration argument. When configuring your Nginx build, you must also add the `--with-pcre-jit` argument.
`pid`	Syntax: File path `pid logs/nginx.pid;` Default value: Defined at compile time. Path of the `pid` file for the Nginx daemon. The default value can be configured at compile time. Make sure to enable this directive, and set its value properly, since the `pid` file may be used by the Nginx `init` script depending on your operating system.

`ssl_engine`	Syntax: Character string
	`ssl_engine enginename;`
	Default value: None
	Where `enginename` is the name of an available hardware SSL accelerator on your system. To check for the available hardware SSL accelerators, run this command from the shell:
	`openssl engine -t`

Name and context	Syntax and description

`thread_pool`	Syntax:
	`thread_pool name threads=number [max_queue=number];`
	Default value:
	`thread_pool default threads=32 max_queue=65536;`
	Defines a thread pool reference that can be used with the `aio` directive for serving larger files asynchronously. Further details are provided in *Chapter 8, Introducing Load Balancing and Optimization*.

`timer_resolution`	Syntax: Numeric (time) `timer_resolution 100ms;` Default value: None Controls the interval between system calls to `gettimeofday()` for synchronizing the internal clock. If this value is not specified, the clock is refreshed after each kernel event notification.

`user`	Syntax:
	`user username groupname;`
	`user username;`
	Default value: Defined at compile time. If still undefined, the user and the group of the Nginx master process are used.
	Allows you to define the user account, and optionally, the user group used for starting the Nginx worker processes. For security reasons, you should make sure to specify a user and a group with limited privileges. For example, create a new user and a group dedicated to Nginx, and remember to apply proper permissions on the files that will be served.

Name and context	Syntax and description

`worker_cpu_affinity`	Syntax:
	`worker_cpu_affinity 1000 0100 0010 0001;`
	`worker_cpu_affinity 10 10 01 01;`
	`worker_cpu_affinity;`
	Default value: None
	This directive works in conjunction with `worker_processes`. It lets you affect the worker processes to CPU cores.
	There are as many series of digit blocks as worker processes; there are as many digits in a block as your CPU has cores.
	If you configure Nginx to use three worker processes, there are three blocks of digits. For a dual-core CPU, each block has two digits:
	`worker_cpu_affinity 01 01 10;`
	The first block (`01`) indicates that the first worker process should be affected to the second core.
	The second block (`01`) indicates that the second worker process should be affected to the second core.
	The third block (`10`) indicates that the third worker process should be affected to the first core.
	Note that affinity is only recommended for multi-core CPUs, not for processors with hyper-treading or similar technologies.

`worker_priority`	Syntax: Numeric `worker_priority 0;` Default value: `0` Defines the priority of the worker processes, from `-20` (highest) to `19` (lowest). The default value is `0`. Note that the kernel processes run at priority level `-5`, so it's not recommended that you set the priority to `-5` or less.

Name and context	Syntax and description
`worker_processes`	Syntax: Numeric or auto `worker_processes 4;` Default value: `1` Defines the number of worker processes. Nginx offers to separate the treatment of requests into multiple processes. The default value is 1, but it's recommended to increase this value if your CPU has more than one core. Besides, if a process gets blocked due to slow I/O operations, the incoming requests can be delegated to the other worker processes. Alternatively, you may use the `auto` value, which will let Nginx select an appropriate value for this directive. By default, it is the amount of CPU cores detected on the system.
`worker_rlimit_core`	Syntax: Numeric (size) `worker_rlimit_core 100m;` Default value: None Defines the size of core files per worker process.
`worker_rlimit_nofile`	Syntax: Numeric `worker_rlimit_nofile 10000;` Default value: None Defines the number of files that a worker process may use simultaneously.
`working_directory`	Syntax: Directory path `working_directory /usr/local/nginx/;` Default value: The prefix switch defined at compile time. A working directory used for worker processes, it is only used to define the location of the core files. The worker process user account (`user` directive) must have write permissions on this folder in order to be able to write core files.
`worker_aio_requests`	Syntax: Numeric `worker_aio_requests 10000;` If you are using `aio` with the `epoll` connection processing method, this directive sets the maximum number of outstanding asynchronous I/O operations for a single worker process.

The Events module

The Events module comes with directives that allow you to configure the network mechanisms. Some of the parameters have an important impact on the application's performance.

All the directives listed in the following table must be placed in the `events` block, which is located at the root of the configuration file:

```
user nginx nginx;
master_process on;
worker_processes 4;
events {
  worker_connections 1024;
  use epoll;
}
[...]
```

These directives cannot be placed elsewhere (if you do so, the configuration test will fail).

Directive name	Syntax and description
`accept_mutex`	Accepted values: `on` or `off` `accept_mutex on;` Default value: `on` Enables or disables the use of an accept `mutex` (mutual exclusion) to open the listening sockets.
`accept_mutex_delay`	Syntax: Numeric (time) `accept_mutex_delay 500ms;` Default value: `500` milliseconds Defines the amount of time that a worker process should wait for before trying to acquire the resource again. This value is not used if the `accept_mutex` directive is set to off.

`debug_connection`	Syntax: IP address or CIDR block.
	`debug_connection 172.63.155.21;`
	`debug_connection 172.63.155.0/24;`
	Default value: None
	Writes detailed logs for clients matching this IP address or address block. The debug information is stored in the file specified with the `error_log` directive, enabled with the debug level.
	Note: Nginx must be compiled with the `--debug` switch in order to enable this feature.

Directive name	Syntax and description
`multi_accept`	Syntax: on or off `multi_accept off;` Default value: `off` Defines whether or not Nginx should accept all the incoming connections at once from the listening queue.
`use`	Accepted values: `/dev/poll, epoll, eventport, kqueue, rtsig`, or `select` `use kqueue;` Default value: Defined at compile time Selects the event model among the available ones (the ones that you enabled at compile time). Nginx automatically selects the most appropriate one, so you should not have to modify this value. The supported models are: • `select`: The default and standard module, it is used if the OS does not support a more efficient one (it's the only available method under Windows). This method is not recommended for servers that expect to be under high load. • `poll`: It is automatically preferred over `select`, but is not available on all systems. • `kqueue`: An efficient method for the FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0, and MacOS X operating systems. • `epoll`: An efficient method for Linux 2.6+ based operating systems. • `rtsig`: Real-time signals, available as of Linux 2.2.19, but unsuited for high-traffic profiles, as the default system settings allow only 1,024 queued signals. • `/dev/poll`: An efficient method for the Solaris 7 11/99+, HP/UX 11.22+, IRIX 6.5.15+, and Tru64 UNIX 5.1A+ operating systems. • `eventport`: An efficient method for Solaris 10, though a security patch is required.
`worker_connections`	Syntax: Numeric `worker_connections 1024;` Default value: None Defines the number of connections that a worker process may treat simultaneously.